Programación de Microservicios con Spring Boot y Red Hat SSO

Nivel Avanzado

Instructor: Carlos Carreño
ccarrenovi@gmail.com

Modulo 8 Autorización y control de acceso por roles

Objetivo: Introducción a Red Hat SSO y su integración con el desarrollo de aplicaciones con Spring Boot

Duración: 1h

Extracción de claims desde el token JWT

Spring Security con OAuth2 / OpenID Connect, puedes extraer las claims de un token JWT de manera sencilla usando el objeto Jwt o el contexto de seguridad (SecurityContext).

continuación

```java
import org.springframework.security.oauth2.jwt.Jwt;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.Map;

@RestController
public class UserController {

    @GetMapping("/user-info")
    public Map<String, Object> getUserInfo(@AuthenticationPrincipal Jwt jwt) {
        return jwt.getClaims(); // Devuelve todas las claims del token JWT
    }
}
```

Uso de anotaciones @PreAuthorize y @RolesAllowed

@PreAuthorize (Spring Security)

✓Permite ejecutar expresiones SpEL para controlar acceso antes de invocar el método.

@RolesAllowed (JSR-250 estándar)

✓Control de acceso basado en roles (sin expresiones SpEL).

Manejo de autorizaciones a nivel de Endpoint y método

- A nivel de Endpoint (vía configuración HTTP Security)

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(authz -> authz
                .requestMatchers("/public/**").permitAll()         // acceso libre
                .requestMatchers("/admin/**").hasRole("ADMIN")     // solo ADMIN
                .requestMatchers("/user/**").hasAnyRole("USER", "ADMIN") // USER o ADMIN
                .anyRequest().authenticated()                      // todo lo demás requiere login
            )
            .oauth2ResourceServer(oauth2 -> oauth2.jwt()); // JWT validation
        return http.build();
    }
}
```

continuación

- A nivel de Método (anotaciones en servicios o controladores)

```java
@RestController
public class DemoController {

    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping("/api/admin-data")
    public String adminData() {
        return "Datos solo para ADMIN";
    }

    @PreAuthorize("#id == authentication.name")
    @GetMapping("/api/user/{id}")
    public String userOwnData(@PathVariable String id) {
        return "Accediendo a datos del usuario: " + id;
    }
}
```

continuación

- on **@RolesAllowed**

```
@RolesAllowed("USER")
@GetMapping("/api/user-data")
public String userData() {
    return "Datos solo para USER";
}
```

Recomendaciones

Escenario	Recomendado
Protección general basada en URL	◆ **Nivel Endpoint (SecurityConfig)**
Lógica de autorización específica por método	◆ **Nivel Método (@PreAuthorize, @RolesAllowed)**
Validaciones complejas (por parámetro, claim)	◆ **Nivel Método con @PreAuthorize (SpEL)**

Lab

- Práctica: Creación de endpoints con control por rol